AndroWatts: Unpacking the Power Consumption of Mobile Device's Components

Édouard Guégain, Rémy Raes, Noé Chachignot, Clément Quinton, Romain Rouvoy

AndroWatts: Unpacking the Power Consumption of Mobile Device's Components

Édouard GUÉGAIN
GREENSPECTOR
eguegain@greenspector.com

Rémy RAES
Inria, Univ. Lille, CNRS,
UMR 9189 CRIStAL
remy.raes@inria.fr

Noé CHACHIGNOT
Inria, Univ. Lille, CNRS,
UMR 9189 CRIStAL
noe.chachignot@inria.fr

Clément QUINTON
Univ. Lille, CNRS, Inria,
UMR 9189 CRIStAL
clement.quinton@univ-lille.fr

Romain ROUVOY
Univ. Lille, CNRS, Inria,
UMR 9189 CRIStAL
romain.rouvoy@univ-lille.fr

***Abstract*—Power efficiency is crucial for mobile devices, where software inefficiencies can rapidly drain battery life and reduce device longevity. To help developers diagnose inefficiency issues and optimize power use, this paper introduces a novel methodology for estimating power consumption at a hardware component level in mobile devices. Unlike existing approaches that rely on coarse-grained battery discharge measurements, our approach consists of modeling per-component power usage as an inverse problem, utilizing linear statistical models and system metrics to estimate the contributions of individual components.**

In this paper, we model the total power usage using system metrics, we evaluate the accuracy of per-component power estimations, and we investigate the impact of dataset size on the model's performance. Our empirical evaluation shows that linear models achieve high predictive accuracy, with R-squared values exceeding 0.90 for total power usage, and robust correlations are observed between predicted and ground-truth power consumption for key components such as the CPU and GPU. We also observe that the amount of measures required to build such a model is of the order of magnitude of the hundred rather than the thousand.

***Index Terms*—Android, component, power, model, battery**

I. INTRODUCTION

In recent years, there has been a growing interest in understanding and optimizing the power consumption of software, particularly in mobile devices [1], [2]. The rapid proliferation of mobile technology and its dependence on finite battery resources have increased the importance of power-efficient design in software development. Users expect longer battery life, and developers strive to meet this expectation without compromising performance. Consequently, accurately measuring and analyzing the power usage of software has become a crucial research area.

A widely adopted method to estimate the power consumption of a specific software component is to monitor the variation in battery discharges over time while the software runs [3]–[5]. Battery-level information is usually accessible via software interfaces (API) that provide a measure of the software's total power consumption. However, this method lacks the precision to attribute power usage to individual hardware components. As a result, developers struggle to identify the root causes of power inefficiencies, eliminating their ability to take targeted corrective actions.

To address this limitation, there is a pressing need for tools capable of modeling the power usage of individual device components. Such models would deliver a more granular analysis of power usage, empowering developers to pinpoint and mitigate power inefficiencies more effectively. This paper explores how system-level metrics, such as system state indicators, can be leveraged to model a device's total power consumption. By building statistical models that estimate the power contributions of individual hardware components based on these metrics, we aim to lay the foundation for more detailed and actionable power consumption analyses. Such a contribution would help developers locate the root cause of the problem more precisely. In particular, this paper addresses the following research questions:

RQ 1: *Can system-level metrics be used to model the total power usage of a device?* The proposed approach uses the coefficients of linear statistical models to estimate the power usage per component. However, these coefficients are only relevant if the model provides qualitative predictions of the overall power usage, making such predictions a necessary precursor to detailed per-component analysis.

RQ 2: *How accurate are per-component power usage estimations?* Since per-component power usage data are derived from modeled data, they inherently carry some degree of inaccuracy. To ensure practical relevance for developers, this inaccuracy must remain within acceptable limits.

RQ 3: *How does the size of the training dataset impact the accuracy of global and detailed power usage estimations?* While larger training datasets are expected to improve the accuracy of both global and detailed power usage estimates, collecting this data is resource-intensive. Additionally, power models for one device may not generalize to others, emphasizing the need to identify the minimal amount of measures required to reach an acceptable level of accuracy.

To answer these research questions, we make the following contributions:

- We examine the possibility of estimating the power drawn

from a device at the battery level, based on a set of metrics representing the state of a device, thus creating a power model.
- We attempt to estimate the power draw of a set of components of a device based on such a model.
- We investigate the effect of reducing the number of measures on the quality of results.
- We share an open-source Android application to artificially put different components of a device under load, an open-source analysis tool to extract metrics from an Android system trace, and a dataset of measures performed with this application and analyzed with this tool[1].

The remainder of the paper is as follows. Section II discusses the related work. Section III introduces the methodology used to gather the dataset studied in this paper. Then, Section IV, Section V, and Section VI explore this dataset to answer the research questions. Section VII discusses the limits of our approach and potential future work.

II. Related Work

The challenge of software power efficiency has seen significant growth in recent years, driven by the pressing environmental crisis, the growing tension on the power grid, and rising public interest in sustainable solutions.

A. Mobile Software Power Efficiency

In particular, the power consumption of mobile software has become a critical research area, as excessive power consumption directly impacts end-users by draining batteries, shortening their lifespan over time, and reducing device usability. Thus, over-consumption can lead to increased environmental waste, as users are more likely to replace hardware more frequently. This creates a compelling incentive for developers to understand and minimize the power usage of their software.

Improving software power efficiency can be achieved by adopting new practices and habits as part of the development processes [6]. Those new routines include:

- Understanding the requirements and goals of the software before the development phase. This involves anticipating optimal software development options [7], [8] and carefully selecting power-efficient third-party libraries, as their power consumption can vary significantly, based on their design [9].
- Testing with environmental impact in mind, on a broader variety of devices to reduce the risk of software obsolescence [10] and identify abnormal consumption patterns during execution.
- Monitoring post-release applications to detect performance variations across environments and address disparities.

However, these power optimizations may conflict with other objectives of the application, such as maintaining performance [11], [12] and security [13], making power efficiency a secondary priority in some cases.

[1]Complementary material:https://zenodo.org/records/14314943

Evaluating the relevance of a power modeling solution requires knowledge of the device's actual power consumption, which serves as the ground truth for validating power model estimations. However, while hardware performance counters are exposed by traditional computer processors [14], they are not included in smartphones' *System-on-Chip* (SoC). Thus, the power usage of mobile devices is often assessed at the battery level, either by integrating a power meter between the battery and the device, or by relying on programmatically available battery charge indicators [15]. However, physical measurement of smartphone power consumption is often impractical due to the increasing trend of non-dismountable design, such as batteries being glued to the casing. Therefore, the overall battery charge is largely used to assess or predict the power efficiency of software, such as mobile applications [16], code snippets [17]–[20], or software design decisions [7], [8], [11], [21]. However, such a metric encompasses the whole power usage of the device and offers only a coarse-grained granularity.

To tackle this limitation, since 2021, some Android devices have been equipped with a per-component power meter, the *On-Device Power Rails Monitor* (ODPM). These power meters are hardware probes placed downstream of the battery, directly before the component. ODPM can monitor the power usage of each of the twelve power rails of the device, with each rail powering one or many components. Available power rails include, for instance, each group of CPU cores, the network components, or the screen. The limitations of this tool are twofold. First, it is only available on a subset of devices.[2] Second, while the metrics are available programmatically from the device, their polling rate is very low, with an update every 30 seconds and an obfuscation is applied to them for security purposes. High polling rates are available when the monitoring is performed through Android Studio, `adb`,[3] or through a system trace, thus limiting measurements to a test environment and making it mandatory to use external tools to analyze the results. Monitoring directly on end-user devices is therefore not practical. Such limitations affect the usability and scalability of ODPM for experiments and lead developers to continue only using the battery data.

However, if the objective is to get a precise consumption per component, adopting ODPM would still be preferable over physical tools.

B. Power Modeling

For mobile devices, the power consumption of a device can be estimated using system metrics such as CPU usage or screen activity [22]. Information such as application logs can also be used to estimate the consumption of specific view components [23]. However, such approaches lack the granularity needed to provide per-component power usage estimations.

[2]Devices of Pixel brand, Pixel 6, and subsequent Pixel devices running Android 10 (API level 29) and higher. New hardware may be released in the future, including by other brands.

[3]https://developer.android.com/tools/adb

Metrics	Description	Unit	Proxy of	Correlation to proxy
`Discharge_rate`	Average battery discharge rate	μW	-	-
24 Power Rails	ODPM metrics for the 24 traced component	μW	-	-
`RedLvl`	Average level of red on the screen during the test	[0-255]	Display	0.16
`GreenLvl`	Average level of green on the screen during the test	[0-255]	Display	0.29
`BlueLvl`	Average level of blue on the screen during the test	[0-255]	Display	0.54
`Brightness`	Brightness of the screen	[1-100]	Display	0.63
`CPU_LITTLE_FREQ`	Average frequency for the little CPU group	KHz	CPU little	0.92
`CPU_MID_FREQ`	Average frequency for the medium CPU group	KHz	CPU mid	0.64
`CPU_BIG_FREQ`	Average frequency for the big CPU group	KHz	CPU big	0.93
`GPU0_FREQ`	Average frequency for the GPU0	KHz	GPU	0.96
`GPU1_FREQ`	Average frequency for the GPU1	KHz	GPU	0.99
`GPU_MEM`	GPU memory usage	Bytes	GPU	0.47
`Wi-Fi_data`	Total of data emitted and received	Bytes	Wi-Fi antenna	0.65

TABLE I: The monitored metrics and their Pearson Correlation Coefficient to the component they represent (All correlations are significant, $p < 0.01$).

Schuler and Kotsis [1] conducted a review of 134 studies on mobile software power consumption published between 2011 and 2021. Their analysis revealed that 80% of these studies relied on a model-based approach, which assumes a correlation between certain variables and the device's power usage. These variables often include metrics from various components, such as the CPU, Display, Wi-Fi, or Cellular modules, and support the definition of the power model. While system metrics are gathered, power data is usually retrieved through battery consumption measurements or via an external tool like the Monsoon Power Monitor [24]. Using these metrics and corresponding power traces, the models are trained to estimate the power consumption share attributable to specific components. However, these strategies present notable limitations:

- The need to model certain components separately due to strong correlations in their power consumption. Ignoring these correlations can result in overestimations.
- The lack of per-component ground-truth measurements, making it difficult to validate the accuracy of the models.

Contrary to such approaches, this paper does not attempt to model the power usage of a specific component. Instead, it adopts a holistic approach, imputing the battery discharge speed to multiple components by simultaneously modeling their respective weight. Furthermore, this paper leverages a ground truth of per-component power usage to validate the accuracy of its prediction, ensuring a more reliable assessment of the model's performance.

Beyond mobile devices, understanding the power usage of software has in particular been studied for imputing power usage to software. Such models attempt to impute a measured or modeled power usage between software components, *e.g.*, between processes running on a processor [25], [26], or between features of a software [27].

III. Experimental Setup

This paper aims to develop a power model for a specific mobile device. To achieve this, we designed a comprehensive experimental setup that generates a dataset of measurements used to train the model. The application[4] used in this setup can stimulate various device features by adjusting parameters, such as the red, green, and blue levels of the screen, the CPU and GPU loads, the brightness of the screen, and the network activity. Stimulating the available range of loads for each component ensures that random scenarios cover a range of workloads as least as large as a real-world usage. In addition, this paper delves into the power consumption of the device in a given workload state, rather than the power consumption of a specific software component running on that device.

Measurements are collected from a structured usage scenario designed to simulate typical device activity. The steps of this scenario are as follows:

1) A random usage profile is generated;
2) The usage profile starts;
3) The system trace record starts;
4) The usage profile runs for 30 seconds;
5) The system trace stops;
6) A screenshot of the usage profile is performed;
7) The usage profile stops.

After each test scenario, data is extracted from the device, including the system trace, the screen brightness, and a screenshot to read the average colors displayed on the screen. The system trace is analyzed using the `PerfettoTraceScript` tool,[5] which aggregates the time series data generated by Android's tracing tool into single values for each monitored metric. Each random scenario is executed only once, as accuracy is expected to emerge from the amount of sampled scenarios rather than from the accuracy of a given scenario. To ensure the reliability of measurements, each test runs for a duration of 30 seconds. However, the first 1.5 seconds of each trace are discarded to compensate for initialization delays and ensure all metrics are fully recorded. The metrics selected for monitoring are designed to act as proxies for various device components, as detailed in Table I, ensuring a comprehensive representation of the device's behavior. While all test scenarios are executed with a fixed duration, data analysis focuses on

[4]https://gitlab.com/greenspector/android-component-stimulator
[5]https://github.com/Zetos11/PerfettoTraceScript

instantaneous values rather than cumulative ones. For example, average discharge speed is used instead of total discharge, and average network speed is used instead of total network data. This approach minimizes the impact of duration variability in the current experiment and facilitates comparability with future studies.

Our empirical experiments were conducted on a Pixel 8 device with build model G9BQD, equipped with a 4575mAh, 4.4V battery, 128GB of storage, a 6.2" 1080x2400 display, 8GB of RAM, and a Google Tensor G3 Nona-core CPU. This CPU is composed of 3 groups of core: The"little" group consisting of 4 cores running at 1.7Ghz, the "middle" group of 4 cores at 2.4Ghz, and the "big" group of 1 core at 2.91Ghz. The device runs Android 15 and includes the ODPM component, providing a ground truth for per-component power usage. To ensure precise control and monitoring, the device is operated via ADB over Wi-Fi, enabling accurate tracking of battery discharge during each test.

IV. Modeling Device Power Usage

The experimental setup described in the previous section enables the creation of a dataset that captures a device's state, such as CPU usage and screen brightness, alongside its power consumption measured through the battery discharge rate. To address the first research question, this section investigates whether the device's overall power consumption can be estimated based on its state alone. Notably, this analysis does not leverage the power usage data for individual components collected during the experiment. In the remainder of this paper, all discussed correlations are calculated using Pearson correlation coefficient, unless specified otherwise.

A. Dataset description

The experimental setup enabled the measurement of $1,000$ random power usage scenarios. The metrics collected during these scenarios are detailed in Table II. Due to inconsistencies in the system tracing tool, some data points were missing—*i.e.*, the frequency of the middle CPU group (15% of cases) and the network usage (12% of cases). To address these gaps and minimize their impact on the overall modeling, missing values were imputed using the mean of their respective data series.

The collected data reveals significant variation across different metrics. For example, screen brightness varied by a factor of 100, the frequency of the medium CPU showed a factor of 150, and the battery discharge speed varies by a factor of 7.1. Such variations indicate that the test scenarios effectively stimulated a wide range of power usage levels on the device. Notably, the battery discharge speed within the dataset exhibited a coefficient of variation of 33.9%.

However, not all metrics contribute to battery discharge speed equally. For instance, CPU-related metrics, particularly `CPU_LITTLE_FREQ`, demonstrated the strongest relationship with battery discharge speed (0.74), followed by the GPU metrics (0.57 and 0.53). In contrast, metrics representing the state of the screen showed a much weaker effect. The average color of the screen is not significantly correlated to the battery discharge speed (0.06, 0.03, and 0.05 for the levels of red, green, and blue respectively, $p > 0.05$). The screen brightness is weakly yet significantly correlated to the battery discharge speed (0.09). This indicates that variations in screen brightness have a comparatively minor impact on battery discharge speed when compared to changes in CPU or GPU frequency. Nonetheless, despite potentially weak correlations to the battery discharge speed, such metrics have stronger correlations to the components they respectively represent. For instance, the brightness of display is correlated to the power usage of the display (0.63).

B. Modeling the battery discharge speed

To predict the battery discharge speed based on the device's state, we compare a set of statistical learning models. The assessed models include Ordinary Least Squares (Linear Regression), Ridge, Lasso, Elastic Net, Decision Tree, Random Forest, Gradient Boosting, and K Neighbors Regressor. All models were trained using 70% of the dataset for training and 30% for testing. Each of these models was trained with optimal hyper-parameters identified from a grid search.[6] The absolute error percentage of each model is reported in Figure 1.

The four linear modeling approaches (LinearRegression, Ridge, Lasso, and Elastic Net) resulted in a very similar performance, achieving an R-squared value of 0.91, indicating that 92% of the variation in the battery discharge speed can be explained by the selected metrics. These models also demonstrated an average absolute error ranging of 8.09%. Outside of linear models, the Random Forest and Gradient Boosting methods performed slightly better, achieving R-squared values of 0.93. These models also had lower average absolute error rates of 6.90% and 6.70%, respectively.

In contrast, the K Neighbors Regressor was less accurate, with an R-squared value of 0.48 and an average absolute error of $17,21\%$. However, this error rate remains acceptable, as the coefficient of variation for the target metric is 33.9%. Thus, even the K Neighbors Regressor substantially outperforms random chance.

> **RQ 1**: The results of this study show that non-power metrics can be effectively leveraged to model a device's total power usage. The linear statistical models demonstrated high predictive performance, achieving strong R-squared values and low average absolute errors relative to the coefficient of variation in battery discharge speed. Non-linear methods, such as Random Forest and Gradient Boosting, offered marginally better predictive accuracy and lower error rates. These findings indicate that the proposed approach produces robust predictions of total power usage, meeting the prerequisite for using linear model coefficients to estimate per-component power contributions. Consequently, this study demonstrates that

[6]The exact parameters are listed in the complementary material.

Metric	mean	std	min	25,00 %	50,00 %	75,00 %	max	Max/min ratio
Discharge_rate	3,60E+06	1,22E+06	1,35E+06	2,74E+06	3,31E+06	4,23E+06	9,63E+06	7,1
RedLvl	122,55	67,25	6,00	67,00	122,00	180,00	241,00	40,2
GreenLvl	122,32	67,64	6,00	64,00	119,00	181,25	241,00	40,2
BlueLvl	124,55	68,17	6,00	66,75	123,00	184,00	241,00	40,2
Brightness	49,13	28,99	1,00	25,00	48,00	74,25	100,00	100
CPU_LITTLE_FREQ	1,23E+06	4,29E+05	7,13E+05	9,45E+05	1,05E+06	1,33E+06	2,49E+06	3,5
CPU_MID_FREQ	1,57E+06	8,22E+05	4,42E+04	8,35E+05	1,49E+06	2,09E+06	6,66E+06	150,7
CPU_BIG_FREQ	1,30E+06	3,76E+05	5,15E+05	1,13E+06	1,15E+06	1,29E+06	2,87E+06	5,6
GPU0_FREQ	5,95E+05	1,79E+05	6,38E+04	5,88E+05	6,21E+05	6,67E+05	8,86E+05	13,9
GPU1_FREQ	5,14E+05	1,78E+05	5,18E+04	4,78E+05	5,30E+05	5,67E+05	8,71E+05	16,8
GPU_MEM	4,97E+08	3,69E+07	3,62E+08	4,72E+08	4,98E+08	5,22E+08	6,13E+08	1,7
Wi-Fi_data	3,76E+06	2,82E+06	7,10E+05	2,18E+06	2,88E+06	4,44E+06	1,76E+07	24,8

TABLE II: The metrics collected with the experimental setup.

Model	MSE	R^2	Min Error (%)	Q1 (%)	Median (%)	Q3 (%)	Max Error (%)	Average Error (%)
Linear Regression	1,28E+11	0,91	0,01	2,92	6,83	11,11	39,69	8,09
Ridge	1,28E+11	0,91	0,01	2,92	6,83	11,11	39,69	8,09
Lasso	1,28E+11	0,91	0,01	2,92	6,83	11,11	39,69	8,09
Elastic Net	1,28E+11	0,91	0,01	2,92	6,83	11,11	39,69	8,09
Decision Tree	2,25E+11	0,84	0,06	3,49	7,71	12,26	51,74	9,35
Random Forest	1,03E+11	0,93	0,03	2,74	5,57	10,31	30,28	6,90
Gradient Boosting	9,61E+10	0,93	0,04	2,82	5,78	9,55	28,52	6,70
KNeighborsRegressor	7,14E+11	0,48	< 0.01	7,10	13,56	23,01	79,27	17,21

TABLE III: The performance of the assessed modeling approaches.

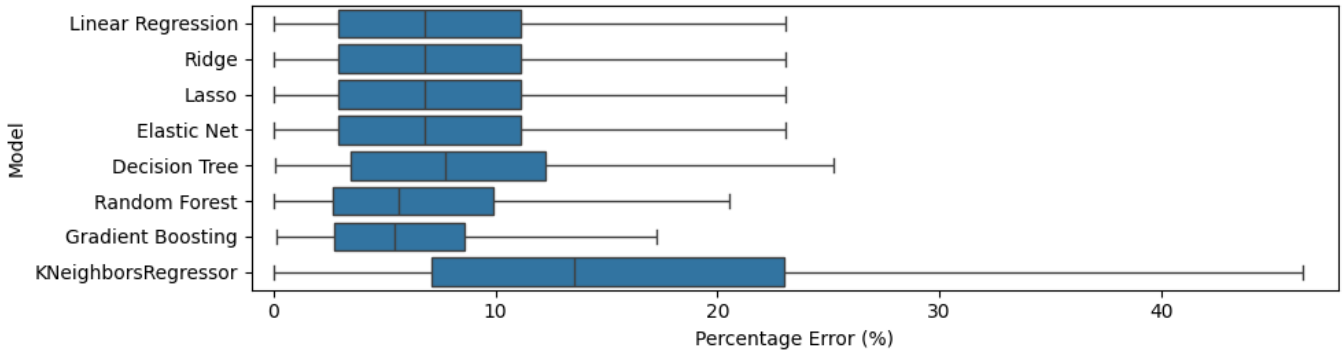


Fig. 1: Error rate of the assessed models.

non-power metrics are a viable and effective means for modeling device power consumption.

V. Modeling per-component power consumption

In linear statistical models, the relationship between a dependent variable (target) and independent variables (features) is represented as a linear combination of the features, weighted by corresponding coefficients. Specifically, the model assumes that the target variable Y can be expressed as shown in Equation 1, where β_0 is the intercept, $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients associated with the features $X_1, X_2, \ldots, X_n$, and ϵ is the error term, accounting for the variability not explained by the linear relationship.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon \quad (1)$$

Given that this model is designed to predict the power drawn by a device, and that the features of this model quantify the activity levels of specific components, then the power drawn by a given component C can be estimated using Equation 2. Here, E_C represents the estimated power draw of the component, while $\beta_1 X_1$ through $\beta_m X_m$ represent a subset of metrics and their respective coefficients that encompass the state of that specific component.

$$E_C = \beta_1 X_a + \beta_2 X_b + \ldots + \beta_p X_m \quad (2)$$

In the context of modeling power usage in mobile devices, the features X_1 through X_n in Equation 1 represent metrics capturing the device's state, and their respective coefficients β_1 through β_n are determined during model training. For instance, in Equation 2, if C represents the display of the device, the features X_1 through X_m would include metrics serving as a proxy for the screen, such as its brightness.

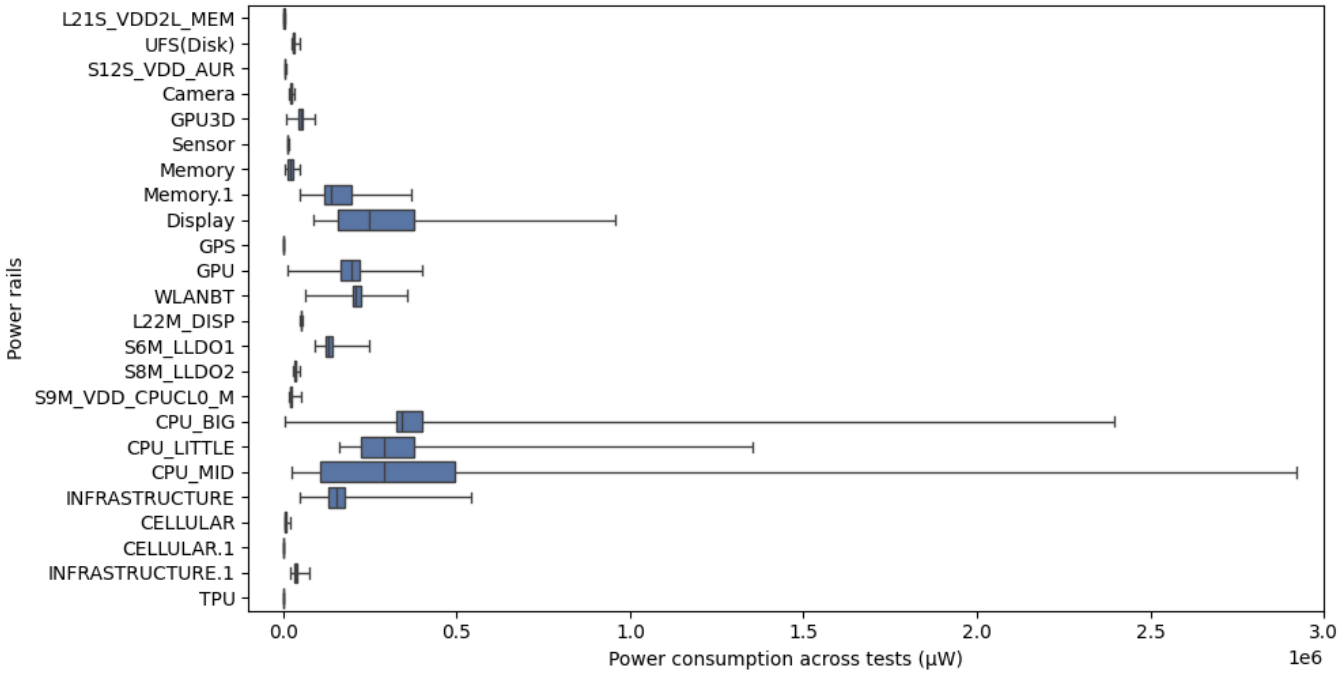


Fig. 2: Measured power draw per component.

Following this formula, this study investigates whether the coefficients of such a model can be used to approximate the power drawn by each component when the device is in a given state.

$$share(E_C) = \frac{\beta_1 X_a + \beta_2 X_b + \ldots + \beta_p X_m}{Y} \quad (3)$$

While such per-component power usage estimations can be validated against the ground truth provided by ODPM, there are some notable limitations when aligning ODPM rails with the estimated and actual battery discharge rates. In particular, the sum of all ODPM rails does not precisely match the actual battery discharge rate. However, as shown in Figure 3, the two metrics exhibit strong correlation (0.97, $p < 0.01$). The sum of power rails is consistently lower than the battery discharge rate by an average factor of 0.73, potentially due to inefficiencies in power conversion from the battery to the rails, or to the existence of components draining power with no associated rail.

This factor can be considered when validating the accuracy of per-component estimations. However, it may differ across devices with ODPM and is unknown for devices without ODPM. To tackle the potential variation between the measured battery discharged rate, the estimated battery discharge rate, and the sum of measured power rails, the power usage of a given component is quantified as a share of the total estimated power usage of the device, rather than an absolute power usage expressed in Watts. Such a value can then be compared to the share of the assessed component group over the sum of all rails.

Equation 3 quantifies the accuracy of the estimation for a component C, denoted as $share(E_C)$. In this equation, the estimated power usage of E_C is quantified with $\beta_1 X_a$ through $\beta_p X_m$, while $share(E_C)$ is calculated as the ratio of this estimated power usage to the predicted total power draw, Y.

A. Ground truth analysis

Among the available power rails, only a subset of them exhibits substantial variation along the test scenarios. The most variable components include the CPU groups (Little, Mid, and Big) and the display. Components such as memory, GPU, Wi-Fi antenna, and "infrastructure" rails show more limited variations while remaining rails (*e.g.*, camera, TPU, cellular antenna) show little to no variations as they were not stimulated during the measure campaign. As a result, the power model does not include features representing these components.

However, the power drawn from certain power rails is strongly correlated using Pearson correlation coefficient. For example:

- The `GPU` and `GPU3D` rails are perfectly correlated (coefficient = 1, $p < 0.01$).
- The `Memory` and `Memory.1` rails are strongly correlated (0.9, $p < 0.01$) and also show high correlations with the `infrastructure` rail (0.89, $p < 0.01$).
- `CPU_MID` and `CPU_LITTLE` rails are also correlated (0.79, $p < 0.01$), with additional correlations to the `Memory.1` rail (respectively 0.81 and 0.69, $p < 0.01$).
- The `infrastructure` rail is correlated with `CPU_BIG` (0.79, $p < 0.01$).

Rail group	Average share (%) (measured)	Average share (%) (estimated)	Average error (%) (relative)	Average error (%) (Absolute)	CV (%) (ground truth)	Correlation (Spearman's ρ)
Display	11,17	7,54	-20,48	29,29	61,48	0,94
CPU & Memory	57,55	57,22	0,83	7,00	16,19	0,96
Wi-Fi	8,92	0,00	-100,00	100,00	31,23	-
GPU	9,24	7,11	-16,44	24,85	34,36	0,92

TABLE IV: Accuracy of per-component predictions (all correlations are significant, $p < 0,01$).

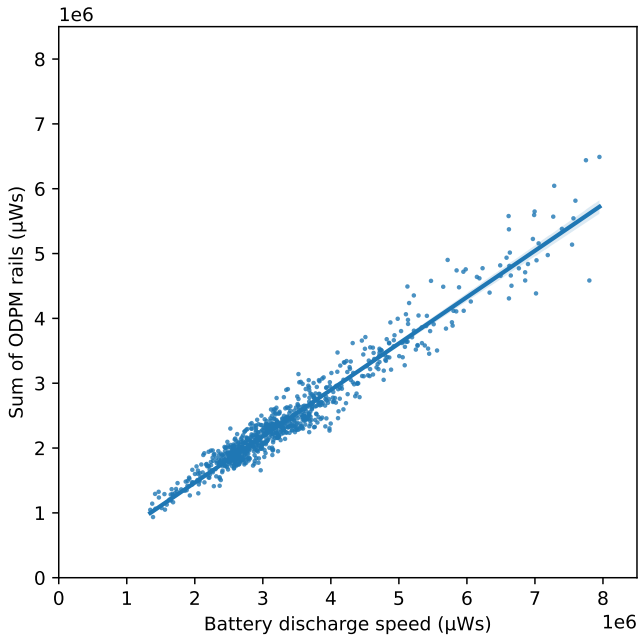


Fig. 3: Correlation between the sum of ODPM rails and the measured battery discharge

Such intricate correlations affect the ability of the model to differentiate the exact power usage for components, like the GPU and GPU3D, as well as the CPU groups (Big, Mid, and Little), the memory, and the infrastructure. Furthermore, the dataset lacks metrics related to memory usage, which could serve as proxies for the memory rail. Consequently, the granularity offered by the power rails may not be fully replicated.

Thus, in the remainder of this paper, the GPU and GPU3D are aggregated into a single indicator: "GPU2/3D". Similarly, the CPU_big, CPU_mid, CPU_little, the memory, and the infrastructure are aggregated into a single indicator named "CPU & Memory". As the rails associated with the display and the Wi-Fi antenna do not show such correlations with others, they are therefore estimated independently.

B. Power rail prediction

In Section IV, all the linear models showed similar performance. However, the approach introduced with Equation 2 introduces specific constraints. As this equation models the power usage of a given component with a subset of metrics and coefficients of the model, negative coefficients may cause the equations to quantify the power usage of a given component as negative. As the components of the smartphone used in the experimental setup are not expected to produce power, the model is forced to use positive coefficients. This causes a slight regression in the overall performance of the model, with a R^2 of 0.87 instead of 0.91 and a median error of 7.52% instead of 6.83%

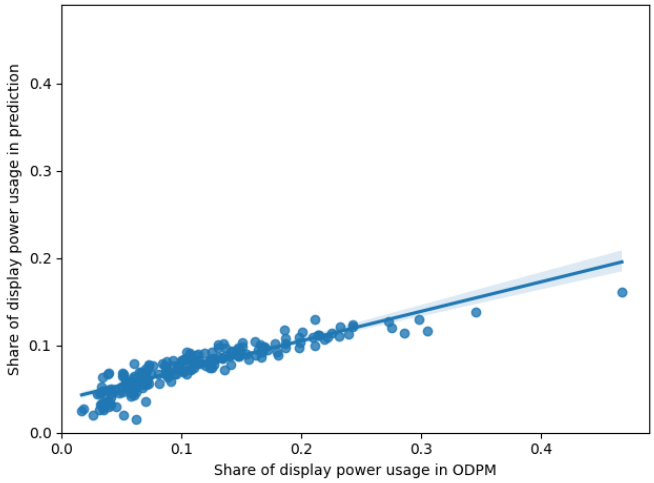
Applying Equation 2 to the best-performing linear model identified in Section IV, using the collected metrics presented in Table I, yields the results presented in Table IV. In that specific table, to compare the predicted and measured share of power, the correlations are computed using Spearman's ρ, in order to compare the order of predictions. In particular:

- The estimated power usage share for the display achieves an absolute error of 29%, while the ground truth for this rail exhibits a coefficient of variation of 61%.
- For the CPU and Memory group, the estimated power usage share has an absolute error of 7%, with a group ground coefficient of variation of 16%.
- The GPU2/3D group achieves an estimated power usage accuracy of 25%, compared to a coefficient of variation of 34% in the ground truth.

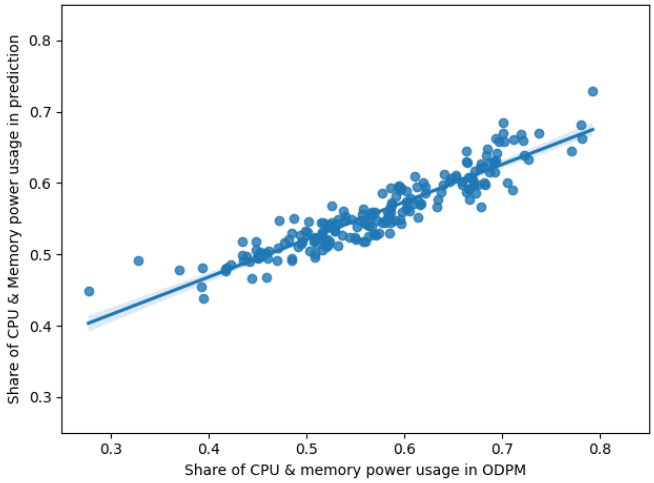
Overall, the model tends to underestimate the power usage of all such rail groups and even fails to predict the power usage of the Wi-Fi rail. The coefficient for the corresponding feature is 0, resulting in predictions of 0 power usage for this rail. Similarly, the `GPU0_FREQ` has a weight of 0 due its redundancy with `GPU1_FREQ`. All other features have non-zero coefficients. Their exact values are listed in Figure 5.

One can observe that the amount of transmitted data and the power usage of the Wi-Fi rail are positively and significantly correlated (0.65, $p < 0.01$), and the rail's power usage is strongly correlated with the device's overall discharge speed. However, the power rail of the Wi-Fi antenna is not significantly correlated to the battery discharge speed (-0.06, $p > 0.01$). Such a behavior is likely due to an interaction with the CPU activity. Indeed, the power rails of the CPU_big and the Wi-Fi antenna are negatively correlated (-0.09, $p < 0.01$). Increased CPU usage may throttle the device, reducing the frequency or amount of network requests. This interaction leads the model to attempt to assign a negative coefficient to the transmitted data metric.

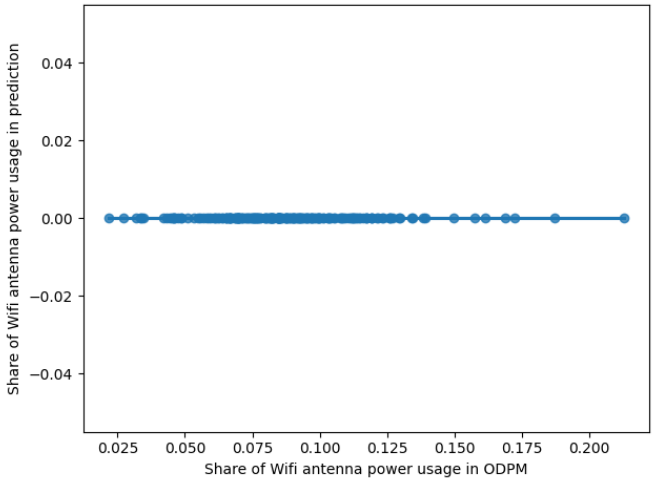
Due to, first, the difference between the battery discharge speed and the sum of available power rail, second, the error rate of the modeled battery discharge speed, and third, the error rate of per-component predictions, quantifying converted the estimated power shares to an absolute power usage expressed



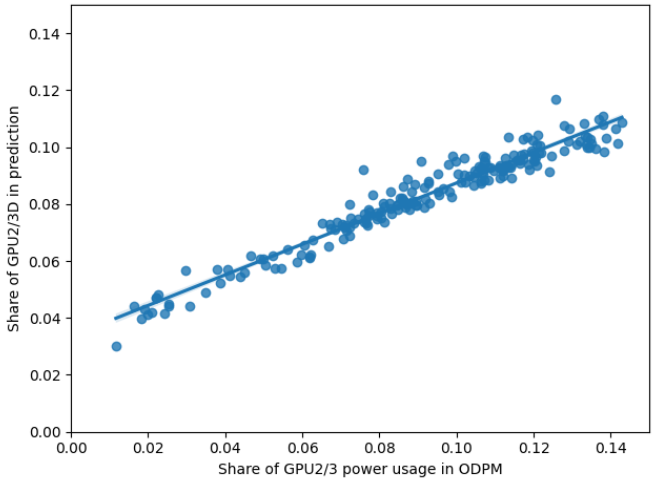
(a) Predicted vs. measured share of the display power usage.



(b) Predicted vs. measured share of the CPU & Memory power usage.



(c) Predicted vs. measured share of the Wi-Fi antenna power usage.



(d) Predicted vs. measured share of the GPU2/3D power usage.

Fig. 4: Comparison of Predicted vs. measured share of power usage for the assessed components groups.

in Watts may not be relevant. Nonetheless, the strong correlations between the estimated and measured power usage of the assessed groups of components can be used to compare different captured states of the devices. For instance, such an approach can be leveraged to compare the power profile of different applications, or different versions of a given application, with a per-component granularity. Reconciling the significant variations in units and power consumption across components would be infeasible without linear modeling. For instance, optimization scenarios that reduce power usage of the `CPU_big` while increasing power consumption of the `CPU_mid` and `CPU_little`, while also affecting the power usage of a component from another group, would require balancing trade-offs. This approach allows for normalizing such variations and can thus be used to explain differences in battery discharge speed with greater granularity.

RQ 2: Linear model coefficients enable the estimation of power usage for individual components of a device, providing strong correlations with the actual power usage of their respective power rails. While the Wi-Fi rail remains an exception due to complex metric interactions, this method achieves a granularity that outperforms battery speed alone, thus allowing the investigation of the root cause of a variation in discharge speed.

VI. Dataset Reduction

The trained model demonstrates high precision when applied to the dataset developed in this experiment. However, this dataset consists of $1,000$ measurements of random usage scenarios. Although each scenario runs for only 30 seconds, there is additional time required for test instrumentation, system trace saving, and trace extraction to the instrumentation terminal. This results in an average runtime of 96 seconds per

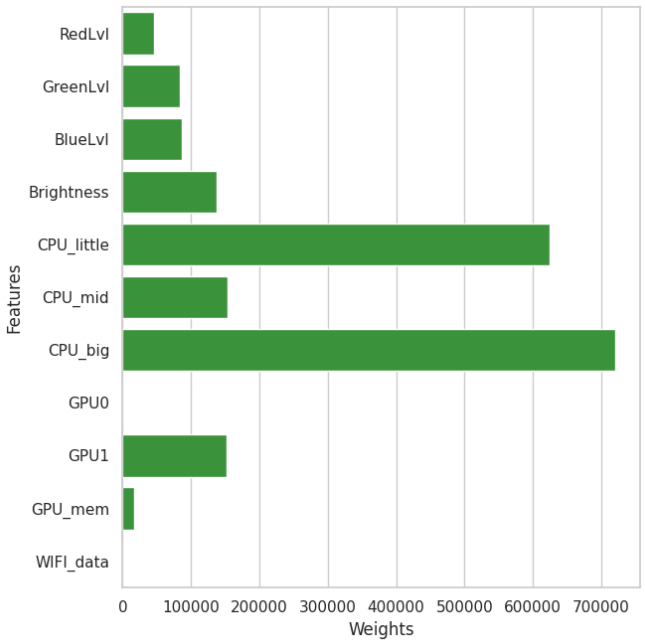


Fig. 5: Weights of the normalized features.

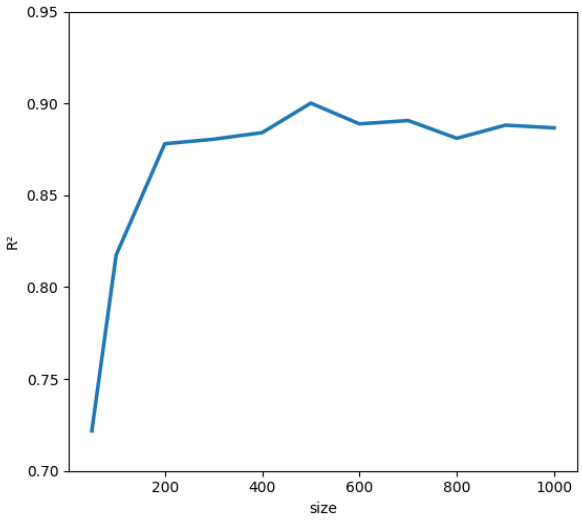


Fig. 6: The evolution of R^2 according to the size of the dataset.

test, and a total runtime of approximately 26.7 hours for the full dataset. Battery limitations add further constraints, as the device needs to recharge every 100 to 150 tests. Additionally, the system traces associated with these tests amount to 256 Gigabytes of data, which require approximately three hours to parse. Given these overheads, replicating this experimental setup for modeling each new device may be inefficient. This section thus explores the impact of dataset size on model precision, in order to identify whether a smaller dataset can maintain comparable performance.

To assess the influence of dataset size, ten subsets of the original dataset were defined, with sizes ranging from 50 to 1000 tests. For each subset, *(i)* a linear model using Elastic Net regularization was trained, *(ii)* optimal parameters were identified via grid search for every sample size, *(iii)* 20% of each subset was reserved for testing, and *(iv)* the reported value is the average of 10 trainings with different random sampled measures. The model accuracies for each sample size are presented in Table V.

We observe that a dataset of 100 measurements or less achieves an R^2 below 0.82. However, from a sample size of 200 onward, R^2 stabilizes at approximately 0.88, with median errors ranging between 6.72 and 7.52. Thus, 200 measures may be the minimum amount to reach a plateau in the capacity of the model to explain the battery discharge speed based on the metrics collected in the experimental setup. Nonetheless, correlations between estimated per-component power usage and their respective rail groups keep increasing to 300 measures, before stabilizing between 0.90 and 0.94 for the Display and CPU & Memory groups. From 500 measures onward, the weight of the transmitted data reaches 0, and the power usage of the Wi-Fi antenna cannot be estimated.

These findings suggest that the number of measurements required for comparable accuracy and stability can be reduced by up to 50%, significantly decreasing the time and storage demands for future experiments. A baseline of 500 measurements appears sufficient for the specific experimental setup in this study, but it may not generalize to other setups where additional variability dimensions, such as TPU or GPS activity, are explored. Thus, while 500 measurements provide a solid starting point, incremental increases should be employed to ensure accuracy plateaus are empirically validated.

> **RQ 3**: The model's precision and the quality of predictions and coefficients quickly converge to a plateau, indicating that additional measurements beyond a certain point do not substantially enhance model performance. While the specific number of measurements needed to reach this plateau may vary with device characteristics and the components analyzed, a baseline of approximately 400 measurements offers an effective starting point for further experimentation.

VII. Discussion

This paper introduces an approach to designing a power model for mobile devices, enabling power usage estimation with per-component granularity. While the results validate the approach, both the experimental setup and statistical modeling introduce some limitations. This section examines these limitations and discusses directions for future work.

*a) **Scope of the experimental setup**:* The validation in this study is limited in scope, as not all device components were assessed. Additional components, such as the GPS, AI accelerator, camera, and other sensors were not stimulated during the experiments, and their power usage was excluded from the model. The studied components were selected based on the availability of data, the ability of developers to affect

Sample size	Battery discharge speed predictions MSE	R^2	Median error (%)	Correlations to measured power usage (Spearman's ρ) Display	CPU & Memory	Wi-Fi	GPU2/3D
50	2,81E+11	0,72	8,91	0,80	0,83	0,63	0,63
100	2,17E+11	0,82	6,53	0,79	0,89	0,53	0,81
200	1,66E+11	0,88	6,72	0,88	0,88	0,03	0,84
300	1,62E+11	0,88	7,52	0,90	0,92	0,48	0,92
400	1,88E+11	0,88	7,46	0,89	0,93	0,35	0,89
500	1,63E+11	0,90	7,19	0,91	0,93	-	0,92
600	1,56E+11	0,89	7,04	0,92	0,94	-	0,93
700	1,72E+11	0,89	6,82	0,92	0,94	-	0,93
800	1,64E+11	0,88	7,42	0,93	0,93	-	0,93
900	1,63E+11	0,89	6,84	0,93	0,94	-	0,93
1000	1,74E+11	0,89	6,81	0,92	0,93	-	0,93

TABLE V: Comparison of sample size of the dataset.

their power consumption, and the fact that such components are always under some workload when the device is running.

Furthermore, only a subset of available metrics was studied to estimate the global and per-component power draw of the device. These metrics were selected as straightforward proxies to estimate the power draw of the chosen components. For instance, the color of the screen is included as it was observed to impact power consumption [28], while the memory usage was excluded as it tends to reduce the accuracy of the model. However, taking into consideration other metrics could potentially enhance the model's accuracy. For instance, device temperature or its variation during a test (both of which are available in the system trace) could contribute to more precise predictions of battery discharge speed. As such metrics are also available with per-component granularity, they may also improve per-component power usage modeling. However, the temperature of the device may also carry information across test scenarios, making it unsuitable for the current experimental design. Allowing the device to cool down between scenarios might mitigate this issue, but would significantly increase the experiment duration.

As described in IV, the system trace data is incomplete for 13% of measures. The system trace format used in this study, introduced with Android 10, is based on the Perfetto format. It remains unclear whether these data holes stem from the on-device tracing tool or from the analysis tools provided by Perfetto. However, the Perfetto project is very active, and ongoing improvements may enhance the quality of system traces. Such improvements would, in turn, improve the quality of the metrics retrieved in this experiment.

*b) **Limitations of per-component predictions**:* As observed in Section V, the total power measured across all device power rails does not match the battery discharge speed, differing by an average ratio of 0.73 in this experimental setup. To the best of our knowledge, there is no empirical evidence to confirm whether this ratio is consistent across devices.

For devices not equipped with ODPM (the primary target of this research), this ratio will be unknown. Thus, converting a component's predicted share of battery discharge speed into an absolute power usage value introduces significant uncertainty. Indeed, the accuracy of such estimates is already constrained by the model's predictive quality and the precision of its coefficients. Adding a potentially incorrect rails-to-battery ratio to the equation would further reduce the reliability of per-components power predictions. Nonetheless, expressing per-component power usage as a relative share of the total estimated power usage remains valuable. This approach facilitates performance comparisons from multiple system traces, such as evaluating different versions of a given application to assess optimizations, while normalizing metrics expressed in different units and monitoring different hardware components.

*c) **Transferability**:* The power model developed in this study is device-specific and cannot be directly transferred to other devices, as differences in hardware configurations, power profiles, and component architectures influence both global and per-component power predictions. For example, the tested device is equipped with nine CPU cores divided into three clusters, each with different maximum frequencies. A different device might have a distinct number of cores, organized into different clusters, operating at different frequencies. Thus, the coefficients identified for the tested device may not be transferable to a different device. While the model itself lacks transferability, the methodology introduced in this paper—*i.e.*, creating and measuring random configurations of load to model per-component power usage—can be adapted to other devices. This includes other smartphones, tablets, computers, or servers. In particular, this approach could be applied to iOS devices, whose power efficiency remains rarely studied due to the closed ecosystem [1]. For devices with similar architectures, such as two smartphones of the same generation, transfer learning could enable partial reuse of the model [29], thus reducing the number of measurements required to achieve the accuracy plateau.

VIII. Conclusion

This paper explores the potential of leveraging device metrics to model its power consumption at the battery level and to estimate per-component power usage, based on the coefficients of such a power model. While the exact power usage of each component cannot be accurately estimated as absolute values, the proposed model provides reliable relative estimates. These relative power usage metrics allow developers to assess power usage variations between different usage scenarios, enabling them to validate optimizations and identify regressions with

per-component granularity. Additionally, such a power model can be built using a limited dataset of randomly generated usage scenarios to stimulate the components of the assessed device.

ACKNOWLEDGMENT

This work is partially funded by the European Union's Horizon 2020 research and innovation program (grant agreement no. GreenDIGIT 101131207) and by the "FEDMALIN" Inria and *Groupe La Poste* partnership. Additionally, this work also received partial support from the French government through the *Agence Nationale de la Recherche* (ANR) under the France 2030 program, including partial funding from the CARECLOUD (ANR-23-PECL-0003) and DISTILLER (ANR-21-CE25-0022), and BPIFRANCE France 2030 CISORANGE grants.

REFERENCES

[1] A. Schuler and G. Kotsis, "A systematic review on techniques and approaches to estimate mobile software energy consumption," *Sustainable Computing: Informatics and Systems*, vol. 41, p. 100919, 2024.

[2] D. Li, S. Hao, J. Gui, and W. G. Halfond, "An empirical study of the energy consumption of android applications," in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 121–130.

[3] P. K. D. Pramanik, N. Sinhababu, B. Mukherjee, S. Padmanaban, A. Maity, B. K. Upadhyaya, J. B. Holm-Nielsen, and P. Choudhury, "Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage," *IEEE Access*, vol. 7, pp. 182 113–182 172, 2019.

[4] J. L. D. Neto, S.-Y. Yu, D. F. Macedo, J. M. S. Nogueira, R. Langar, and S. Secci, "Uloof: A user level online offloading framework for mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 11, pp. 2660–2674, 2018.

[5] L. Corral, A. B. Georgiev, A. Sillitti, and G. Succi, "A method for characterizing energy consumption in android smartphones," in *2013 2nd International Workshop on Green and Sustainable Software (GREENS)*, 2013, pp. 38–45.

[6] E. Guégain, "Assessing the environmental impact of mobile applications: a measure framework toward devgreenops," in *Proceedings of the IEEE/ACM 11th International Conference on Mobile Software Engineering and Systems*, 2024, pp. 88–91.

[7] R. Horn, A. Lahnaoui, E. Reinoso, S. Peng, V. Isakov, T. Islam, and I. Malavolta, "Native vs web apps: Comparing the energy consumption and performance of android apps and their web counterparts," in *2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2023, pp. 44–54.

[8] V. Frattaroli, O. Le Goaer, and O. Philippot, "Ecological impact of native versus cross-platform mobile apps: a preliminary study," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 2023, pp. 3–8.

[9] X. Zhan, T. Liu, L. Fan, L. Li, S. Chen, X. Luo, and Y. Liu, "Research on third-party libraries in android apps: A taxonomy and systematic literature review," *IEEE Transactions on Software Engineering*, vol. 48, no. 10, pp. 4181–4213, 2022.

[10] L. Mosesso, N. Maudet, E. Nano, T. Thibault, and A. Tabard, "Obsolescence paths: Living with aging devices," in *2023 International Conference on ICT for Sustainability (ICT4S)*, 2023, pp. 13–23.

[11] A. Bogdan and I. Malavolta, "An empirical study on the impact of css prefixes on the energy consumption and performance of mobile web apps," in *Proceedings of the IEEE/ACM 11th International Conference on Mobile Software Engineering and Systems*, ser. MOBILESoft '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 12–21. [Online]. Available: https://doi.org/10.1145/3647632.3647989

[12] W. Oliveira, R. Oliveira, and F. Castor, "A study on the energy consumption of android app development approaches," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 42–52.

[13] J. Ferreira, B. Santos, W. Oliveira, N. Antunes, B. Cabral, and J. P. Fernandes, "On security and energy efficiency in android smartphones," in *2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2023, pp. 87–95.

[14] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: memory power estimation and capping," in *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 189–194. [Online]. Available: https://doi.org/10.1145/1840845.1840883

[15] B. Dornauer and M. Felderer, "Energy-saving strategies for mobile web apps and their measurement: Results from a decade of research," in *2023 IEEE/ACM 10th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 2023, pp. 75–86.

[16] É. Guégain, T. Simon, A. Rahier, and R. Rouvoy, "Managing uncertainties in ict services life cycle assessment using fuzzy logic," in *International Conference on ICT for Sustainability (ICT4S)*. IEEE, 2024.

[17] O. Le Goaer and J. Hertout, "Ecocode: A sonarqube plugin to remove energy smells from android projects," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–4.

[18] H. Anwar, D. Pfahl, and S. N. Srirama, "Evaluating the impact of code smell refactoring on the energy consumption of android applications," in *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2019, pp. 82–86.

[19] S. Hao, D. Li, W. G. Halfond, and R. Govindan, "Estimating mobile application energy consumption using program analysis," in *2013 35th international conference on software engineering (ICSE)*. IEEE, 2013, pp. 92–101.

[20] D. Li, S. Hao, W. G. Halfond, and R. Govindan, "Calculating source line level energy information for android applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, 2013, pp. 78–89.

[21] V. M. F. Jacques, N. Alizadeh, and F. Castor, "A study on the battery usage of deep learning frameworks on ios devices," in *Proceedings of the IEEE/ACM 11th International Conference on Mobile Software Engineering and Systems*, ser. MOBILESoft '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1–11. [Online]. Available: https://doi.org/10.1145/3647632.3647990

[22] L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. M. Mao, Z. Wang, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2010, pp. 105–114.

[23] H. Furusho, K. Hisazumi, T. Kamiyama, H. Inamura, T. Nakanishi, and A. Fukuda, "Poster: an energy profiler for android applications used in the real world," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 517–518. [Online]. Available: https://doi.org/10.1145/2307636.2307712

[24] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "Software-based energy profiling of android apps: Simple, efficient and reliable?" in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 103–114.

[25] G. Fieni, D. R. Acero, P. Rust, and R. Rouvoy, "PowerAPI: A Python framework for building software-defined power meters," *Journal of Open Source Software*, vol. 9, no. 98, p. 6670, Jun. 2024. [Online]. Available: https://hal.science/hal-04601379

[26] "Scaphandre on github," https://github.com/hubblo-org/Scaphandre, accessed: 2024-12-08.

[27] É. Guégain, C. Quinton, and R. Rouvoy, "On reducing the energy consumption of software product lines," in *Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume A*, 2021, pp. 89–99.

[28] "Should you switch your wallpaper to affect less the battery life of your smartphone?" https://greenspector.com/en/should-you-switch-your-wallpaper-to-affect-less-the-battery-life-of-your-smartphone/, accessed: 2024-12-08.

[29] D. Obst, B. Ghattas, S. Claudel, J. Cugliari, Y. Goude, and G. Oppenheim, "Improved linear regression prediction by transfer learning," *Computational Statistics & Data Analysis*, vol. 174, p. 107499, 2022.